



AF  
ifw  
PATENTS

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE  
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

APPELLANT: Gschwind DOCKET: YO999-357 (8728-320)  
SERIAL NO.: 09/492,544 GROUP ART UNIT: 2183  
FILED: January 27, 2000 EXAMINER: Meonske, Tonia L.  
FOR: **METHODS FOR RENAMING A MEMORY REFERENCE TO STACK  
LOCATIONS IN A COMPUTER PROCESSING SYSTEM**

Mail Stop Appeal Brief-Patents  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

**RESPONSE TO NOTICE OF NON-COMPLIANCE WITH APPEAL BRIEF**

In response to the Notification of Non-Compliant Appeal Brief dated 12/21/04 and the Final Office Action dated March 24, 2004 finally rejecting Claims 1, 3-31 and 33-39 under 35 U.S.C. §102(b) and 35 U.S.C. 103(a). Applicant appeals pursuant to the Notice of Appeal filed on August 5, 2004 and submit this appeal brief.

---

**CERTIFICATE OF MAILING UNDER 37 C.F.R. §1.8(a)**

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on January 21, 2005.

Dated: January 21, 2005

  
Nathaniel T. Wallace



## TABLE OF CONTENTS

	<u>Page</u>
1. REAL PARTY IN INTEREST	1
2. RELATED APPEALS AND INTERFERENCES	1
3. STATUS OF CLAIMS	1
4. STATUS OF AMENDMENTS	1
5. SUMMARY OF CLAIMED SUBJECT MATTER	2
6. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL	4
7. ARGUMENT	4
A.    The Claim Rejections Under 35 U.S.C. §102 Are Legally Deficient	5
i.    Claims 12 and 26	5
B.    The Claim Rejections Under 35 U.S.C. §103 Are Legally Deficient	6
i.    Claims 1 and 31	6
C.    Conclusion	9
8. CLAIMS APPENDIX	10
9. EVIDENCE APPENDIX	NONE
10. RELATED PROCEEDINGS APPENDIX	NONE



**1. Real Party in Interest**

The real party in interest is International Business Machines Corporation, the assignee of the entire right, title, and interest in and to the subject application by virtue of an assignment of record.

**2. Related Appeals and Interferences**

None.

**3. Status of Claims**

Claims 1, 3-31 and 33-39 are pending, stand rejected, and are under appeal.

Claims 2 and 32 have been cancelled.

A copy of the Claims as pending is presented in the Appendix.

**4. Status of Amendments**

Claims 9, 12, 18, 26 and 39 were amended by Amendment under 37 C.F.R. §1.111 filed April 23, 2004. This Amendment was entered.

Claims 1, 3, 31 and 33 were amended and Claims 2 and 32 were cancelled by Amendment under 37 C.F.R. §1.116 filed May 24, 2004. This Amendment was entered.

## **5. Summary of Claimed Subject Matter**

The present invention relates to methods for renaming stack references in a computer processing system. The stack references are renamed to processor-internal registers. By concentrating on the frequent rename opportunities for stack references a renaming architecture can be efficiently implemented.

References to the processor stack use a limited number of stack-management registers. This reduces the possible ambiguities that can arise in the renaming of memory locations using different general-purpose registers. It is determined whether an instruction references a location in a local stack of a processor using an architecturally defined register for accessing the location. A reference to the stack location of the processor by any means other than the stack-management registers results in performing consistency-preserving actions. The consistency-preserving operations include synchronizing an architected state between processor-internal registers and a main memory of the computer processing system. By maintaining a consistency main memory state, out-of-order execution of memory operations may be performed.

Referring to claim 1; a method is claimed for renaming memory references to stack locations in a computer processing system. The method includes detecting stack references that use architecturally defined stack access methods as described, for example, at page 18, lines 13-14 and in Figure 2, element 212. The method includes replacing the stack references with references to processor-internal registers as described at, for example, page 21, lines 13-16 and in Figure 3, element 322. Similar disclosure is included with respect to Figures 4 and 5, elements 422 and 522, respectively. The method further includes synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system, as described at, for example, page 32, lines 6-12.

Referring to claim 12; a method is claimed for renaming memory reference to stack locations in a multiprocessor computer processing system. The method includes determining whether a load instruction references a location in a local stack using an architecturally defined register for accessing a stack location, wherein each processor comprises a respective local stack, as described, for example, at page 19, lines 13-16 and in Figure 3, element 310. The method claims determining whether a rename register exists for the references location in the local stack, when the load instruction references the location using the architecturally defined register, as described at page 21, lines 7-12, and in Figure 3, element 320. The method further includes replacing the reference to the location by a reference to the rename register, when the rename register exists, as described at page 21, lines 13-16, and in Figure 3, element 322.

Referring to claim 26; a method is claimed for renaming memory reference to stack locations in a multiprocessor computer processing system. The method includes determining whether a store instruction references a location in a local stack using an architecturally defined register for accessing a stack location, wherein each processor comprises a respective local stack, as described at page 26, lines 6-8, and in Figure 5, element 510. The method claims allocating a rename register for the location, when the store instruction references the location using the architecturally defined register, as described at page 27, lines 20-21, and in Figure 5, element 520. The method further includes replacing the reference to the location by a reference to the rename register, as described at page 28, lines 5-6, and in Figure 5, element 522.

Referring to claim 31; claim 31 includes substantially the limitations of claim 1. References to the specification therefore include the references given for claim 1. Claim 31 is embodied in a program storage device readable by machine, for example, as described as page 30, lines 15-20 and Figure 6.

**6. Grounds of Rejection to be Reviewed on Appeal**

A. Claims 1, 3-11, 31 and 33-39 stand rejected under 35 U.S.C. 103 as being obvious in view of the combined teachings of Katzman et al. (U.S. Patent No. 3,737,871) and Morris (U.S. Patent No. 6,286,095).

B. Claims 12-24 and 26-30 stand rejected under 35 U.S.C. 102 as being anticipated by Katzman et al. (U.S. Patent No. 3,737,871).

**7. Argument**

**A. The Claim Rejections Under 35 U.S.C. 102 Are Legally Deficient.**

Under 35 U.S.C. §102, a claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference. The identical invention must be shown in as complete detail as is contained in the claim. See MPEP §2131.

**i. Claims 12 and 26**

It is respectfully submitted that at the very least, Katzman is legally deficient to establish a case of anticipation against independent Claims 12 and 26.

Katzman fails to teach the claimed device including “determining whether a load instruction references a location in a local stack using an architecturally defined register for accessing a stack location, wherein each processor comprises a respective local stack” as claimed in Claim 12 and “determining whether a store instruction references a location in a local stack using an architecturally defined register for accessing a stack location, wherein each processor comprises a respective local stack” as claimed in Claim 26.

The Examiner has relied upon inherency to reject claims 12 and 26. Specifically, as provided in the Advisory Action, the Examiner stated, “in order for Katzman’s operations to reference the stack, the operations must inherently determine to reference the stack.”

“In relying upon the theory of inherency, the examiner must provide a basis in fact and/or technical reasoning to reasonably support the determination that the allegedly inherent characteristic necessarily flows from the teachings of the applied prior art.” *Ex parte Levy*, 17 USPQ2d 1461, 1464 (Bd. Pat. App. & Inter. 1990) (Emphasis added). See also MPEP §2112. To establish inherency, the extrinsic evidence must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. *In re Robertson*, 169 F.3d 743, 745, 49 USPQ2d 1949, 1950-51 (Fed. Cir. 1999) (Emphasis added). See also MPEP §2112.1(IV).

Katzman operates stack registers using “specialized stack operations” (see col. 4, lines 31-35). The specialized stack operations operate on the stack by definition. Thus, Katzman does not teach, inherently or otherwise, “determining whether a load instruction references a location in a local stack using an architecturally defined register for accessing a stack location”, as claimed in Claim 12, or “determining whether a store instruction references a location in a local stack”, as claimed in Claim 26 (Emphasis added). Because Katzman’s operations are known by definition to reference the stack, there is no determination, inherent or otherwise, of whether the operations reference the stack. Katzman’s operations reference only the stack. Therefore, it is not necessary to determine whether the operations reference a location in a local stack; it is a given, there is no alternative for a stack operation but to reference the stack. Nowhere does Katzman teach or suggest any determination with respect to whether an instruction references the

stack, essentially as claimed in Claims 12 and 26. Therefore, Katzman fails to teach all the limitations of claims 12 and 26.

Accordingly, the rejection of Claims 12-30 should be overruled.

**B. The Claim Rejections Under 35 U.S.C. 103 Are Legally Deficient.**

In rejecting claims under 35 U.S.C. §103, the Examiner bears the initial burden of presenting a *prima facie* case of obviousness. In re Rijckaert, 9 F.3d 1531, 1532 (Fed. Cir. 1993). The burden of presenting a *prima facie* case of obviousness is only satisfied by showing some objective teaching in the prior art or that knowledge generally available to one of ordinary skill in the art would lead that individual to combine the relevant teachings of the references. In re Fine, 837 F.2d 1071, 1074 (Fed. Cir. 1988). A *prima facie* case of obviousness is established when the teachings of the prior art itself would appear to have suggested the claimed subject matter to one of ordinary skill in the art. In re Bell, 991 F.2d 781, 782 (Fed. Cir. 1993). If the Examiner fails to establish a *prima facie* case, the rejection is improper and must be overturned. In re Rijckaert, 9 F.3d at 1532 (citing In re Fine, 837 F.2d at 1074).

**i. Claims 1 and 31**

It is respectfully submitted that at the very least, the combined teachings of Katzman and Morris are legally deficient to establish a *prima facie* case of obviousness against independent Claims 1 and 31.

The combined teachings of Katzman and Morris fail to teach or suggest the claimed device including “synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system” as claimed in Claims 1 and 31.



Claims 1 and 31 claim, *inter alia*, “synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system.”

The combined teachings of Katzman and Morris are legally deficient to establish a *prima facie* case of obviousness against Claim 1 or Claim 31 because Katzman and Morris do not teach or suggest “synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system,” as claimed in Claims 1 and 31.

Katzman teaches a plurality of top-of-stack registers for a stack management system with dedicated operations (see col. 1 lines 62-65, and col. 4, lines 31-35). Stack registers are used for storing a number of stack registers filled with stack information, storing a number representing a naming states, and storing the location of the top piece of information (see Abstract). The registers of Katzman are not processor-internal registers, essentially as claimed in claims 1 and 31. The registers of Katzman exist in main memory (see col. 2, line 68 to col. 3, line 2). The registers of Katzman are not processor-internal registers. Further, nowhere does Katzman teach or suggest another memory having a synchronized state with the stack registers. Therefore, Katzman does not teach or suggest synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system, essentially as claimed in claims 1 and 31. Therefore, Katzman fails to teach or suggest all the limitations of claims 1 and 31.

Morris teaches forced load and store operations (see col. 5, lines 21-24). The forced load and store operations prevent out-of-order processing, thus avoiding mis-synchronization of CPUs (see col. 6, lines 58-61). Morris does not teach or suggest, “synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system” as claimed in claims 1 and 31. Morris teaches that the load and store operations can be executed

without synchronization problems between CPUs (see col. 7, lines 5-10). Morris teaches preventing mis-synchronization between CPUs through the use of blocking, suspending all subsequent load operations until a prior operation is completed (see col. 5, lines 46-48). Morris' method of avoiding mis-synchronization of CPUs through blocking is not analogous to synchronizing an architected state between the processor-internal registers and a main memory, essentially as claimed in claims 1 and 31. Morris suspends operations to prevent CPU mis-synchronization. Morris does not teach or suggest a step of "synchronizing." Preventing mis-synchronization is clearly distinct from synchronizing. Thus, Morris fails to teach or suggest, "synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system" as claimed in claims 1 and 31. Therefore, Morris fails to cure the deficiencies of Katzman.

The combined teachings of Katzman and Morris fail to teach or suggest "synchronizing", much less "synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system" as claimed in claims 1 and 31. The Examiner's reconsideration of the rejection is respectfully requested.

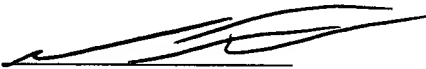
Because the combined teachings of Katzman and Morris fail to teach or suggest every limitation of Claims 1 and 31, it is respectfully asserted that no case of anticipation has been made out.

Claims 3-11 depend from Claim 1. Claims 33-39 depend from Claim 31. The dependent claims include the elements of their respective independent claims and they are not rendered unpatentable by the cited references for at least the reasons given for the independent claims.

### **C. CONCLUSION**

The claimed invention is not disclosed or suggested by the teachings of the applied prior art references, either alone or in combination. Moreover, the Examiner has failed to establish a case of anticipation of the presently claimed method under 35 U.S.C. §102 over Katzman with respect to Claim 12 and 26 for at least the reasons noted above. Further, the Examiner has failed to establish a *prima facie* case of obviousness of the presently claimed method under 35 U.S.C. §103 over Katzman and Morris with respect to Claims 1 and 31, for at least the reasons noted above. Accordingly, it is respectfully requested that the Board overrule the rejection of Claims 1-3, 5, 6, 9-11 and 16-21 under 35 U.S.C. §102 and 35 U.S.C. §103.

Date: Jan 21 '05

By:   
Nathaniel T. Wallace  
Reg. No. 48,909  
Attorney for Appellants

**F. CHAU & ASSOCIATES, LLP**  
130 Woodbury Road  
Woodbury, New York 11797  
TEL: (516) 692-8888  
FAX: (516) 692-8889

## **8. CLAIMS APPENDIX**

What is claimed is:

1. A method for renaming memory references to stack locations in a computer processing system, comprising the steps of:

detecting stack references that use architecturally defined stack access methods;

replacing the stack references with references to processor-internal registers; and

synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system.

3. The program storage device according to claim 1, wherein said synchronizing step comprises the step of inserting in-order write operations for all of the stack references that are write stack references.

4. The method according to claim 1, further comprising the step of performing a consistency-preserving operation for a stack reference that does not use the architecturally defined stack access method.

5. The method according to claim 4, wherein said step of performing a consistency-preserving operation comprises the step of bypassing a value from a given processor-internal register to a load operation that references a stack area and that does not use the architecturally defined stack access methods.

6. The method according to claim 4, further comprising the step of synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system, and wherein said step of performing a consistency-preserving operation comprises the step of recovering an in-order value for the stack reference from the main memory, upon performing said synchronizing step.

7. The method according to claim 6, wherein the in-order value is written to the main memory by an in-order write operation inserted into an instruction stream containing an instruction corresponding to the stack reference, when the stack reference is a write stack reference.

8. The method according to claim 6, further comprising the step of writing the in-order value to the main memory in response to a load operation that does not use the architecturally defined stack access methods.

9. The method according to claim 1, wherein said step of performing a consistency-preserving operation comprises the steps of:

discarding all out-of-order state;

synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system; and

restarting execution after a store operation has been performed that does not use the architecturally defined stack access methods.

10. The method according to claim 1, wherein the architecturally defined stack access methods comprise memory accesses that use at least one of a stack pointer, a frame pointer, and an argument pointer.
11. The method according to claim 1, wherein the architecturally defined stack access methods comprise push, pop, and other stack manipulation operations.
12. A method for renaming memory reference to stack locations in a multiprocessor computer processing system, comprising the steps of:
- determining whether a load instruction references a location in a local stack using an architecturally defined register for accessing a stack location, wherein each processor comprises a respective local stack;
  - determining whether a rename register exists for the references location in the local stack, when the load instruction references the location using the architecturally defined register; and
  - replacing the reference to the location by a reference to the rename register, when the rename register exists.
13. The method according to claim 12, wherein the architecturally defined register corresponds to a pointer for accessing stack locations.
14. The method according to claim 13, wherein the pointer for accessing the stack locations is one of a stack pointer, a frame pointer, and an argument pointer.

15. The method according to claim 12, wherein the architecturally defined stack access methods comprise push, pop, and other stack manipulation operations.
16. The method according to claim 12, wherein said step of determining whether the renaming register exists comprises the step of computing one of a symbolic address and an actual address of the location.
17. The method according to claim 12, wherein said step of determining whether the rename register exists is performed during one of a decode, and address generation, and a memory access phase.
18. The method according to claim 12, further comprising the steps of:  
determining that the rename register does not exist; and  
performing the load instruction from one of a main memory and a cache of the system.
19. The method according to claim 12, further comprising the step of determining whether the load instruction references a location in any stack, including the local stack, using another register, when the load instruction does not reference the location using the architecturally defined register.
20. The method according to claim 19, wherein said step of determining whether the load instruction references the location using the other register comprises the step of marking

translation lookaside buffer (TLB) entries of pages in the local stack as containing stack references, when the load instruction references the location using the other register.

21. The method according to claim 19, further comprising the step of performing the load instruction from one of a main memory and a cache of the system, when the load instruction does not reference the location using the other register.

22. The method according to claim 19, further comprising the step of executing a consistency-preserving mechanism to perform the load instruction from the stack area, when the load instruction references the location using the other register.

23. The method according to claim 12, further comprising the step of allocating a rename register for the location, when the rename register does not exist.

24. The method according to claim 23, further comprising the step of inserting an operation, into an instruction stream containing the load instruction, to load the location from a processor of the system to the allocated rename register, upon allocating the rename register.

25. The method according to claim 24, further comprising the step of replacing the reference to the location by a reference to the allocated rename register, upon inserting the operation.

26. A method for renaming memory reference to stack locations in a multiprocessor computer processing system, comprising the steps of:



determining whether a store instruction references a location in a local stack using an architecturally defined register for accessing a stack location, wherein each processor comprises a respective local stack;

allocating a rename register for the location, when the store instruction references the location using the architecturally defined register; and

replacing the reference to the location by a reference to the rename register.

27. The method according to claim 26, further comprising the step of inserting an operation, into an instruction stream containing the store instruction, to store the location from the rename register to a main memory of the system, upon replacing the reference to the location by the reference to the rename register.

28. The method according to claim 26, further comprising the step of determining whether the store instruction references a location in any stack, including the local stack, using another register, when the store instruction does not reference the location using the architecturally defined register.

29. The method according to claim 28, further comprising the step of performing the store instruction from one of a main memory and a cache of the system, when the store instruction does not reference the location using the other register.

30. The method according to claim 28, further comprising the step of executing a consistency-preserving mechanism to perform the store instruction to the stack area, when the store instruction references the location using the other register.

31. A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform methods steps for renaming memory references to stack locations in a computer processing system, the method steps comprising:

- detecting stack references that use architecturally defined stack access methods;
- replacing the stack references with references to processor-internal registers; and
- synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system.

33. The program storage device according to claim 31, wherein said synchronizing step comprises the step of inserting in-order write operations for all of the stack references that are write stack references.

34. The program storage device according to claim 31, further comprising the step of performing a consistency-preserving operation for a stack reference that does not use the architecturally defined stack access method.

35. The program storage device according to claim 34, wherein said step of performing a consistency-preserving operation comprises the step of bypassing a value from a given

processor-internal register to a load operation that references a stack area and that does not use the architecturally defined stack access methods.

36. The program storage device according to claim 34, further comprising the step of synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system, and wherein said step of performing a consistency-preserving operation comprises the step of recovering an in-order value for the stack reference from the main memory, upon performing said synchronizing step.

37. The program storage device according to claim 36, wherein the in-order value is written to the main memory by an in-order write operation inserted into an instruction stream containing an instruction corresponding to the stack reference, when the stack reference is a write stack reference.

38. The program storage device according to claim 36, further comprising the step of writing the in-order value to the main memory in response to a load operation that does not use the architecturally defined stack access methods.

39. The program storage device according to claim 34, wherein said step of performing a consistency-preserving operation comprises the steps of:

discarding all out-of-order state;

synchronizing an architected state between the processor-internal registers and a main memory of the computer processing system; and

restarting execution after a store operation has been performed that does not use the architecturally defined stack access methods.